Bringing Introspection Into the BlobSeer Data-Management System Using the MonALISA Distributed Monitoring Framework

Alexandra Carpen-Amarie*, Jing Cai†, Alexandru Costan‡, Gabriel Antoniu*, Luc Bougé§
* *Centre Rennes - Bretagne Atlantique, INRIA/IRISA, France*
{alexandra.carpen-amarie,gabriel.antoniu}@inria.fr
† *Department of Computer Science, City University of Hong Kong*
Tylor.Cai@student.cityu.edu.hk
‡ *Department of Computer Science, University Politehnica of Bucharest*
Alexandru.Costan@cs.pub.ro
§ *ENS Cachan/Brittany, IRISA, Rennes, France*
Luc.Bouge@bretagne.ens-cachan.fr

***Abstract*—Introspection is the prerequisite of an autonomic behavior, the first step towards a performance improvement and a resource-usage optimization for large-scale distributed systems. In grid environments, the task of observing the application behavior is assigned to monitoring systems. However, most of them are designed to provide general resource information and do not consider specific information for higher-level services. More specifically, in the context of data-intensive applications, a specific introspection layer is required in order to collect data about the usage of storage resources, about data access patterns, etc. This paper discusses the requirements for an introspection layer in a data-management system for large-scale distributed infrastructures. We focus on the case of BlobSeer, a large-scale distributed system for storing massive data. The paper explains why and how to enhance BlobSeer with introspective capabilities and proposes a three-layered architecture relying on the MonALISA monitoring framework. This approach has been evaluated on the Grid'5000 testbed, with experiments that prove the feasibility of generating relevant information related to the state and the behavior of the system.**

***Keywords*-Distributed system, storage management, large-scale system, monitoring, introspection.**

I. Introduction

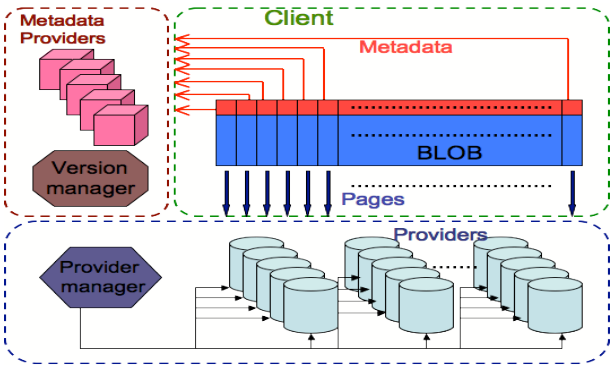
In the area of distributed systems, introspection mechanisms play a crucial role in assisting the users in overcoming the challenges raised by the behavior of their systems at large scales. First, introspection relies on monitoring tools, which provide the users with the feedback necessary for identifying the state of their application and the state of the infrastructure where the application is running on, at a particular moment in time. Second, the monitored information can further be fed back into the system and used by self-managing engines, in order to enable an autonomic behavior of the system [7] [12], possibly with several goals such as self-configuration, self-optimization, or self-healing.

On existing geographically-distributed platforms (e.g. grids), introspection is often limited to low-level tools for monitoring the physical nodes and the communication interconnect: they typically provide information such as CPU load, network traffic, job status, file transfer status, etc. In general, such low-level monitoring tools focus on gathering and storing monitored data in a scalable and non-intrusive manner [14].

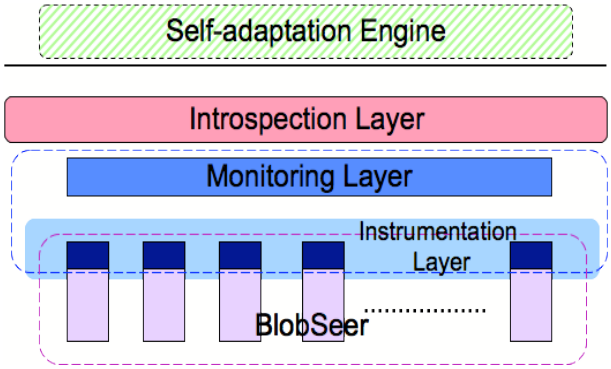
Even though many grid monitoring applications have been developed to address such general needs [9] [5], little has been done when it comes to enabling introspection for large-scale distributed data management. This is particularly important in the context of data-intensive applications distributed at a large scale. In such a context, specific parameters related to data storage need to be monitored and analyzed in order to enable self-optimization in terms of resource usage and global performance. Such parameters regard physical data distribution, storage space availability, data access patterns, application-level throughput, etc.

This paper discusses the requirements of a large-scale distributed data-management service in terms of introspection: it explains in which specific ways introspection can serve to enable an autonomic behavior of the data-management service. As a case study, we focus on BlobSeer [10], a service for sharing massive data at very large scales in a multi-user environment. We propose a three-layered architecture enabling BlobSeer with introspection capabilities. We validate our approach through an implementation based on the generic MonALISA [8] monitoring framework for large-scale distributed services.

The remainder of the paper is organized as follows. Section II provides a brief description of BlobSeer. Section III explains which self-adaptation goals can be served by introspection mechanisms in such a data-management system and which data need to be col-



(a) The architecture of the BlobSeer system



(b) The architecture of the introspective BlobSeer

Figure 1. BlobSeer

lected. It also describes the solution we designed and implemented, based on the MonALISA monitoring framework. Section IV illustrates the usefulness of our approach by discussing some monitoring experiments realized on the Grid'5000 testbed. Finally, Section V draws conclusions and directions for future developments.

II. BlobSeer

BlobSeer is a data-sharing system which addresses the problem of efficiently storing massive, unstructured data blocks called *binary large objects* (referred to as BLOBs further in this paper), in large-scale, distributed environments. The BLOBs are fragmented into small, equally-sized chunks, called *pages*. BlobSeer provides an efficient fine-grained access to the pages belonging to each BLOB, as well as the possibility to modify them, in distributed, multi-user environments.

The architecture of BlobSeer (Figure 1(a)) includes multiple, distributed entities. *Clients* initiate all BLOB operations: CREATE, READ, WRITE and APPEND. There can be many concurrent clients accessing the same BLOB or different BLOBs in the same time. The support for concurrent operations is enhanced by storing the pages belonging to the same BLOB on multiple *storage providers*. The metadata associated with each BLOB are hosted on other components, called *metadata providers*. BlobSeer provides versioning support, so as to prevent pages from being overwritten and to be able to handle highly-concurrent WRITE and APPEND operations. For each of them, only a patch composed of the range of written pages is added to the system. Finally, the system comprises two more entities: the *version manager* that deals with the serialization of the concurrent WRITE/APPEND requests and with the assignment of version numbers for each new WRITE/APPEND operation; the *provider manager*, which keeps track of all storage providers in the system.

As far as this paper is concerned, an APPEND operation can be considered as a special case of WRITE. Therefore, we disregard this distinction in the rest of the paper. Everything stated about WRITEs is also true for APPENDs, unless explicitly specified.

A typical setting of the BlobSeer system involves the deployment of a few hundreds storage providers, storing BLOBs in the order of the TB. The typical size for a page within a blob can be smaller that 1 MB, whence the challenge of dealing with hundreds of thousands of pages belonging to just one BLOB. BlobSeer provides efficient support for heavily-concurrent accesses to the stored data, reaching a throughput of 6.7 GB/s aggregated bandwidth for a configuration with 60 metadata providers, 90 data providers and 360 concurrent writers, as explained in [11].

III. Towards an introspective BlobSeer

Our goal is to enhance BlobSeer with introspection capabilities, in order to enable this data-sharing platform with an autonomic behavior. To meet this goal, we have designed a three-layered architecture aiming at identifying and generating relevant information related to the state and the behavior of the system (Figure 1(b)). Such information is then expected to serve as an input to a higher-level *self-adaptation* engine (currently not implemented yet). These data are yielded by an (1) *introspection* layer, which processes the raw data collected by a (2) *monitoring* layer. The lowest layer is represented by the (3) *instrumentation* code that enables BlobSeer to send monitoring data to the upper layers.

A. Self-adaptation: what to adapt?

To introduce an autonomic behavior in BlobSeer, we considered several aspects.

Dynamic dimensioning: Extensive performance evaluations [11] carried out for BlobSeer reveal that the aggregate bandwidth of concurrent WRITE or READ operations grows as the number of data providers and metadata providers increases. However, deploying BlobSeer's providers on a large number of physical nodes can be an expensive approach, and their optimum number is often unpredictable, as it depends on the load of the providers and on the number of clients concurrently accessing them.

These aspects justify the need for a mechanism enabling a dynamic adjustment of the number of running data/metadata providers, according to the state of the system and its real-time requirements in terms of load and number of available nodes.

State-dependent allocation algorithms for storage providers: Currently, each time a client writes some data on a BLOB, it receives a set of providers from the provider manager, and writes each page on one of them. The provider manager allocates the providers in a round-robin manner; therefore a balanced storage-space load among providers is expected.

Nevertheless, the behavior may deviate from the expected one, as several factors can impact the efficiency of such a straightforward load-balancing algorithm. The number of concurrent clients requesting access to the same provider plays a major role in the performance of the WRITE or READ operations. Concurrent data accesses have to be serialized, leading to slow or even unresponsive providers. A more advanced provider-allocation algorithm taking into account such factors that influence the behavior of the providers would bring an improvement of the overall performance of BlobSeer.

Adaptive data replication strategies: BlobSeer is designed to be used by data-intensive applications, such as the ones related to astronomy, data mining or multimedia processing. To fully fit the needs of these applications, it also has to ensure that a replication degree is maintained for the stored data. When a BLOB is created, the client has to specify the number of replicas that will be generated for each of its pages. But the "optimal" number of replicas may vary across the BLOB's versions and even across pages, as their usage patterns may be different. As a consequence, BlobSeer can benefit from a self-tuning mechanism for dynamic selection of the replication level for each BLOB.

B. Introspection: what data to collect?

All the improvement directions stated above can only be effective if the self-adaptation engine receives accurate data from the **introspection layer**. The latter generates data ranging from general information about the running nodes to specific data regarding the stored BLOBs and their structure.

General information: These data are essentially concerned with the physical resources of the nodes that act as storage providers. They include CPU usage, network traffic, disk usage, storage space or memory.

A self-adapting system has to take into account information about the values of these parameters across the nodes that make up the system, as well as about the state of the entire system. For instance, the used and available storage space at each single provider play a crucial role in deciding whether additional providers are needed or not. Besides the values for these basic data belonging to each provider, the system also needs access to aggregated data, such as the value of the total storage space occupied/available for the entire system.

Individual BLOB-related data: The most significant information for a single BLOB is its access pattern, i.e. the way the pages and the versions are accessed through READ and WRITE operations. The basic data are the number of read accesses for each page that the BLOB version consists of, and the number of write operations performed on the BLOB for each page. Since each WRITE or READ operation consists in accessing a range of consecutive pages, it is expected that some ranges of pages will have the same number of accesses. As a consequence, these data facilitate the identification of the regions of the BLOB comprising pages with the same number of accesses, information that can influence the adopted replication strategy.

From another viewpoint, the number of accesses can be associated with the version of the BLOB that they refer to. This approach enables a comparison between the sizes of the versions and their usage, i.e. the number of READ requests for each of them. This is a valuable information for the replication algorithms, which can assign more replicas to the versions highly accessed by the clients.

Global state: Even though the provider-allocation algorithm or the replication strategy have access to the details within each BLOB, it is not irrelevant to have an overview of the whole data stored in the BlobSeer system, from a higher-level point of view. Some of the key data at this global level are the total number of accesses associated with each provider. This is a measure of the load of each of them and can directly influence the selection of the providers that will be allocated new pages, depending on their deviation from the average load within the system.

The other system-wide data refer to the distribution of all the BLOBs across providers. The number of BLOB slices hosted on each provider, as well as their sizes, comprise a compact information about the way the data are managed. It can trigger a response from

the provider manager, in case there are load variations between the providers with respect to one or more BLOBs. It can be equally useful to expose the BLOBs that have a high rate of change or growth, as opposed to the BLOBs that contain data that is seldom modified. The dynamic growth of the BLOBs can be emphasized through the number of WRITE operations performed on each BLOB, which is equivalent to the number of versions, or through the number of pages written for each BLOB, i.e. the total size of its versions.

C. Monitoring: how to collect?

The input for the introspective layer consists of raw data that are extracted from the running nodes of BlobSeer, collected and then stored, a set of operations realized within the **monitoring layer**. Therefore, it can rely on a monitoring system designed for large-scale environments that implements these features. Such a monitoring framework has to be both scalable and extensible, so as to be able to deal with the huge number of events generated by a large-scale data-management system, as well as to accommodate system-specific monitoring information and to offer a flexible storage schema for the collected data.

The monitoring framework – MonALISA: The Global Grid Forum [4] proposed a Grid Monitoring Architecture (GMA) [13], which defines the components needed by a scalable and flexible grid monitoring system: producers, consumers, and a directory service. A wide variety of grid monitoring systems [14], such as Ganglia [9], RGMA [2], GridICE [1], comply with this architecture.

Among them, we selected MonALISA (*Monitoring Agents in a Large Integrated Services Architecture*) [8] for our data-monitoring tasks, as it is a general-purpose, flexible framework, which provides the necessary tools for collecting and processing monitoring information in large-scale distributed systems. Moreover, it is an easily-extensible system, which allows the definition and processing of user-specific data, by means of an API for dynamically-loadable modules. MonALISA is currently used to monitor large high-energy physics facilities; it is deployed on over 300 sites belonging to several experiments, such as CMS or ALICE [3].

MonALISA is based on four layers of services, the *Lookup and Discovery Services*, the *MonALISA services* – the components dealing with the data-collection tasks, the *Proxy services* that make possible the communication between the services and the clients and the *MonALISA clients and repositories*, which act as consumers.

The main challenge the monitoring layer has to cope with is the large number of storage provider nodes and therefore the huge number of BLOB pages, versions and huge BLOB sizes. Furthermore, it has to deal with hundreds of clients that concurrently access various parts of the stored BLOBs, as they generate a piece of monitoring information for each page accessed on each provider. MonALISA is suitable for this task, as it is a system designed for large-scale environments and it proved to be both scalable and reliable.

Instrumenting BlobSeer: The data generated by the **instrumentation layer** are relayed by the monitoring system and finally fed to the introspection layer. The instrumentation layer is implemented as a component of the monitoring layer. The MonALISA framework provides a library called ApMon that can be used to send the monitoring data to the MonALISA services. At the providers, the instrumentation code consists in listeners located on each of them, which report to the monitoring system each time a page is written or read. The monitoring information from the version manager is collected using a parser that monitors the events recorded in the logs. The state of the physical resources on each node is monitored through an ApMon thread that periodically sends data to the monitoring service.

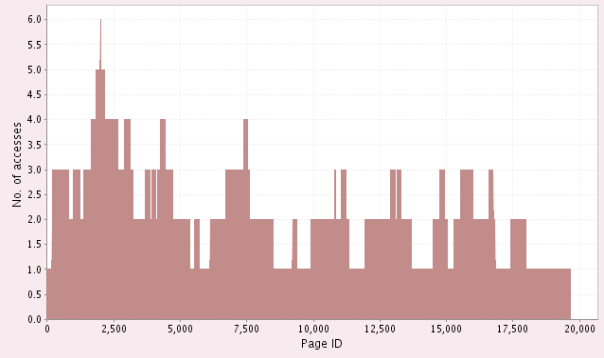
IV. Illustration

A. Experimental plan

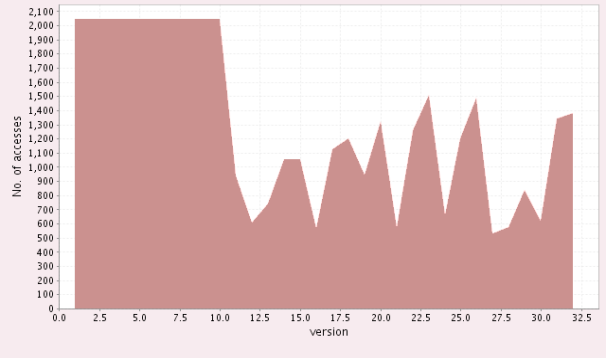
We evaluated the feasibility of gathering and interpreting the BlobSeer-specific data needed to ensure input data for the different self-optimizing directions. Our approach was to create an introspection layer on top of the monitoring system, able to process the raw data collected from BlobSeer and to extract significant information regarding the state and the behavior of the system. We tested it on the Grid'5000 [6] testbed, a large-scale experimental grid platform, with reconfiguration and control capabilities, that covers 9 sites geographically distributed in France.

For the experiments, we used 127 nodes belonging to a Grid'5000 cluster in Rennes. The nodes are equipped with x86_64 CPUs and at least 4 GB of RAM. They are interconnected through a Gigabit Ethernet network. We deployed each BlobSeer entity on a dedicated node, as follows: two nodes were used for the version manager and the provider manager, 10 nodes for the metadata providers, 100 nodes for the storage providers and 10 nodes acted as BlobSeer clients, writing data to the BlobSeer system. Four nodes hosted MonALISA services, which transferred the data generated by the instrumentation layer built on top of the BlobSeer nodes to a MonALISA repository. The repository is the location where the data were stored and made available to the introspection layer.

In this experiment, we used 10 BLOBs, each of them having the page size of 1 MB and a total size larger than 20 GB. We started the 10 clients, each of them having to create a BLOB and to write 10 data



(a) Number of WRITE accesses on the logical address pages for a BLOB



(b) Version access patterns

Figure 2. BLOB accesses visualization

blocks of 2 GB each on it. Each data block overlaps the previous one by 10%. Next, we started the clients in parallel and each of them performed a number of WRITE operations on a randomly selected BLOB. The blocks were written on the BLOB at random offsets and they consisted of a random number of pages, ranging between 512 MB and 2 GB in size. This experiment lasted for a dozen of minutes. All figures below are real graphical representations of data provided by the introspection layer at the end of this experiment.

B. Results

We processed the raw data collected by the monitoring layer and extracted the higher-level data within the introspection layer. Some results are presented below, along with some graphical representations.

Storage Data: The most straightforward data that can be obtained from the collected parameters refers to the physical resources. Two of them are the available/used storage space, which are as well one of the most significant information concerning the data providers. There are two approaches to deal with the information about the storage space.

The first one is exposing a *global view*, which depicts the current values of the available and used storage space for the entire system, as a sum of all the most recent free and used space amounts reported by the storage providers. This is a real-time measure of the system load, indicating whether the system was overloaded, or in a stable and safe state.

The second approach aims at providing a *detailed view*. It describes the occupied and available space at each storage provider, as opposed to the global view that presented the load of the whole system.

Access patterns: They represent a significant information that the introspection layer has to be aware of. It can be obtained by computing the number of

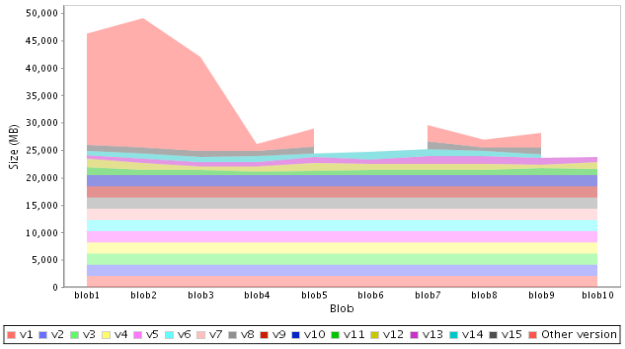


Figure 3. BLOBs versions and sizes

READ/WRITE accesses. The access patterns can be examined from two points of view. The first one regards the access patterns for each BLOB. It considers the number of read accesses for each page, for a specified version or for the whole BLOB and it identifies the regions of the BLOB composed of pages with the same number of accesses (Figure 2(a)). The other one refers to the number of READ or WRITE operations performed on each provider, allowing a classification of the providers according to the pressure of the concurrent accesses they have to withstand.

The distribution of READ/WRITE accesses across the BLOB versions: The versions can be weighted by counting the number of read accesses or of physical memory pages for each of them, as shown in Figure 2(b). This approach exposes the tendencies of the READ operations over the versions or the various sizes of the WRITE operations, as the number of write accesses for a specified version is equivalent with the number of pages written.

The distribution of the BLOBs across providers: It exposes the total sizes of the stored BLOBs, as well as the way they are stored on the providers, providing

a comprehensive image of how the system manages the stored data. This information is used to detect the variations of BLOBs' loads across providers.

The structure and sizes of all the stored BLOBs: The differences between BLOBs, in terms of size or number of versions, highlight the BLOBs that have the most important growth (Figure 3). This information, along with the number of accesses for each BLOB, emphasize the most valuable BLOBs in the system.

V. Conclusion

This paper addresses the challenges raised by the introduction of introspection into a data-management system for large-scale, distributed infrastructures. Such a feature aims at exposing general and service-specific data to a higher-level layer, in order to enable the system to evolve towards an autonomic behavior. We propose a layered architecture built on top of the BlobSeer data-management system, a service dedicated to large-scale sharing of massive data. The goal of this architecture is to generate a set of specific data that can serve as input for a self-adaptive engine. The architecture consists of 3 layers: 1) an instrumentation layer that extracts the low-level, raw data from the different components of BlobSeer; 2) a monitoring layer that deals with collecting and storing the monitoring data from the instrumentation layer and 3) an introspective layer that processes the gathered data into higher-level information describing the state and the behavior of the system.

To build the monitoring layer, we relied on the MonALISA general-purpose, large-scale monitoring framework, for its versatility and extensibility. Our experiments showed that it was able to scale with the number of BlobSeer providers and to cope with the huge amount of monitoring data generated by a large number of clients. Moreover, it allowed us to define and to collect BlobSeer-specific data, and to extend the existing visualization charts with new ones that met the requirements of BlobSeer. The experiments performed confirm the outcome of the introspection layer, by means of graphical representations associated with the various high-level data extracted.

The next step will consist in equipping BlobSeer with a self-adaptive engine that will employ the output of the introspection layer to optimize its performance and resource usage. As an example, by allowing BlobSeer to dynamically dimension the number of each of its entities, this engine can help improving the storage resource allocation strategy. Besides, it can also provide information based on which adaptive data replication strategies can be implemented. Together, such features will enable an autonomic behavior of the BlobSeer data-management platform.

Acknowledgments

The authors thank Bogdan Nicolae for his crucial technical support regarding BlobSeer. The experiments reported in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see http://www.grid5000.org/).

References

[1] S. Andreozzia, N. De Bortoli, S. Fantinel, et al. GridICE: a monitoring service for grid systems. *Future Generation Computer Systems*, 21(4):559–571, April 2005.

[2] A. Cooke, A. Gray, W. Nutt, et al. The relational grid monitoring architecture: Mediating information about the grid. *Journal of Grid Computing*, 2(4):323–339, 2004.

[3] The MonALISA Repository for ALICE. http://pcalimonitor.cern.ch/map.jsp.

[4] Global Grid Forum. http://www.ggf.org/.

[5] D. Gunter et al. Netlogger: A toolkit for distributed system performance analysis. *Intl. Symp. on Modeling, Analysis, and Simulation of Computer Syst.*, page 267, 2000.

[6] Y. Jégou et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *Intl. Journal of High Performance Comp. Applications*, 20(4):481–494, 2006.

[7] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[8] I. Legrand, H. Newman, R. Voicu, et al. MonALISA: An agent based, dynamic service system to monitor, control and optimize grid based applications. In *Computing for High Energy Physics*, Interlaken, Switzerland, 2004.

[9] M. Massie, B. Chun, and D. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[10] B. Nicolae, G. Antoniu, and L. Bougé. BlobSeer: How to enable efficient versioning for large object storage under heavy access concurrency. In *Data Management in Peer-to-Peer Systems*, St-Petersburg, Russia, 2009.

[11] B. Nicolae, G. Antoniu, and L. Bougé. Enabling high data throughput in desktop grids through decentralized data and metadata management: The BlobSeer approach. In *Proceedings of the 15th International Euro-Par Conference*, pages 404–416, Delft, Netherlands, 2009.

[12] Manish Parashar and Salim Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*, pages 247–259. Springer Verlag, 2005.

[13] B. Tierney, R. Aydt, D. Gunter, et al. A grid monitoring architecture. Grid Working Draft GWD-PERF-16-3, August 2002. http://www.gridforum.org/.

[14] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computing Systems*, 21(1):163–188, 2005.